

SKIRT: an Advanced Dust Radiative Transfer Code with a User-Friendly Architecture

Peter Camps*, Maarten Baes

Sterrenkundig Observatorium, Universiteit Gent, Krijgslaan 281, B-9000 Gent, Belgium

Abstract

We discuss the architecture and design principles that underpin the latest version of SKIRT, a state-of-the-art open source code for simulating continuum radiation transfer in dusty astrophysical systems, such as spiral galaxies and accretion disks. SKIRT employs the Monte Carlo technique to emulate the relevant physical processes including scattering, absorption and emission by the dust. The code features a wealth of built-in geometries, radiation source spectra, dust characterizations, dust grids, and detectors, in addition to various mechanisms for importing snapshots generated by hydrodynamical simulations. The configuration for a particular simulation is defined at run-time through a user-friendly interface suitable for both occasional and power users. These capabilities are enabled by careful C++ code design. The programming interfaces between components are well defined and narrow. Adding a new feature is usually as simple as adding another class; the user interface automatically adjusts to allow configuring the new options. We argue that many scientific codes, like SKIRT, can benefit from careful object-oriented design and from a friendly user interface, even if it is not a graphical user interface.

Keywords: radiative transfer, numerical methods, dust, object-oriented design, abstraction, modularity

1. Introduction

The presence of even a small fraction of dust can have a substantial impact on the radiation traversing and exiting an astrophysical system. In a typical spiral galaxy viewed edge-on, for example, the central dust lane blocks most of the starlight in the UV and optical wavelength range and re-emits the absorbed energy in the infrared and sub-millimeter regime (e.g. Verstappen et al. 2013). Simulating the precise effect of the dust is not trivial. Anisotropic scattering by the dust couples all lines of sight, and dust absorption/emission couples all wavelengths. As a result, the radiative transfer equation is highly nonlocal and nonlinear (Steinacker et al. 2013). In most astrophysical systems, at least part of the dust is not in local thermal equilibrium with the radiation field, complicating the calculations even more (Draine and Li 2001). And finally, realistic scenarios involve complex 3D geometries such as spiral arms, randomly placed clumps, or snapshots taken from a hydrodynamical simulation.

Because of these complexities, most dust radiative transfer codes use the Monte Carlo technique to tackle the problem; see e.g. the reviews by Whitney (2011) and Steinacker et al. (2013). The radiation field is represented as a stream of discrete photon packages. A simulation follows the individual path of each photon package through the dusty medium. The trajectory is governed by various events determined statistically by drawing random numbers from the appropriate probability distribution. Typically, a photon package is emitted, undergoes a number of scattering events, and is finally either absorbed or leaves the

system. The Monte Carlo technique is conceptually simple and allows efficient radiative transfer calculations for complex problems. However, due to the randomization process, the results inherently contain a certain level of Poisson noise.

SKIRT is a state-of-the-art Monte Carlo dust radiative transfer code. It implements the common optimization techniques, such as peel-off at emission and scattering events (Yusef-Zadeh et al. 1984), continuous absorption (Lucy 1999; Niccolini et al. 2003), and forced scattering (Cashwell and Everett 1959), and includes novel techniques such as the library mechanism described in Baes et al. (2011). The code is registered in the Astrophysics Source Code Library with identifier *ascl:1109.003*. Earlier versions were described in Baes et al. (2003) and in Baes et al. (2011). Here we present the latest, substantially revised version of SKIRT, which is fully documented¹ and publicly available from a GitHub code repository².

Stalevski et al. (2012) have used SKIRT to investigate the emission of active galactic nuclei (AGN) dusty tori in the infrared domain, modeling the dusty torus as a two-phase medium with high-density clumps and a low-density medium filling the space between the clumps. The resulting SED database has been made public as described in Stalevski (2012). As can be expected, SKIRT is often used to solve the *inverse* radiative transfer problem, where the goal is to recover the actual 3D distribution of radiation sources and dust by fitting the results of radiative transfer simulations to observational data. This is a nontrivial task, since the underlying model typically has a large number of free parameters. For example, SKIRT has been used

*Corresponding author

Email address: peter.camps@ugent.be (Peter Camps)

¹SKIRT documentation: <http://www.skirt.ugent.be>

²SKIRT code repository: <https://github.com/skirt/skirt>

to perform detailed studies of the dust energy balance in the edge-on spiral galaxies UGC4754, NGC 4565, and M104 (Baes et al. 2010; De Looze et al. 2012a,b). These studies found an inconsistency in the dust energy budget, suggesting that a sizeable fraction of the total dust reservoir consists of a clumpy distribution with no associated young stellar sources.

Rather than using a manual trial and error procedure, De Geyter et al. (2013) describe FitSKIRT, a code that automatically fits a 3D model to observed images of a dusty galaxy by matching the output of SKIRT radiative transfer simulations to the data. They apply FitSKIRT to automatically determine the intrinsic distribution of stars and dust in the galaxy NGC 4013. De Geyter et al. (2014) use FitSKIRT to investigate interstellar dust in a sample of 12 edge-on galaxies, simultaneously reproducing the g -, r -, i - and z -band observations from a model with 19 free parameters without human intervention.

In this article, we do not discuss the results obtained with SKIRT, nor the Monte Carlo radiative transfer techniques implemented in the code. Instead we focus on the software design choices involved with setting up the simulation model. Many scientific codes require a user to hard-code the model makeup for each distinct problem. In contrast, our strategy with SKIRT in recent years has been to continuously add new features without removing existing capabilities. Consequently, SKIRT now offers a wealth of configurable components that are ready to use without any programming at all, especially in the areas where the code has been most often applied. An ad-hoc approach to including all of this functionality would have lead to source code that is hard to understand, maintain, and use. Instead we developed a modular, generic software architecture that can support the wide range of built-in components and options in SKIRT in a developer- and user-friendly way.

In Sect. 2 we first provide an overview of SKIRT’s features, including the user interface for configuring a particular simulation. In Sect. 3 we then discuss the design goals for the latest revision of the code, we describe the overall architecture, and we zoom in on a few key aspects of the design, such as the mechanism that automatically adjusts the user interface to accommodate new features. In Sect. 4 we finally argue that many scientific codes, like SKIRT, can benefit from careful object-oriented design and from a friendly user interface, even if it is not a graphical user interface.

2. Features

2.1. Overview

SKIRT is a Monte Carlo continuum radiative transfer code for simulating the effect of dust on radiation in static astrophysical systems. It is assumed that the radiation traverses the system much faster than the time scale on which the system evolves. SKIRT offers full treatment of absorption and multiple anisotropic scattering by the dust, computes the temperature distribution of the dust and the thermal dust re-emission self-consistently, and supports stochastic heating of small grains using an efficient library approach. It handles multiple dust mixtures and arbitrary 3D geometries for radiation sources and dust

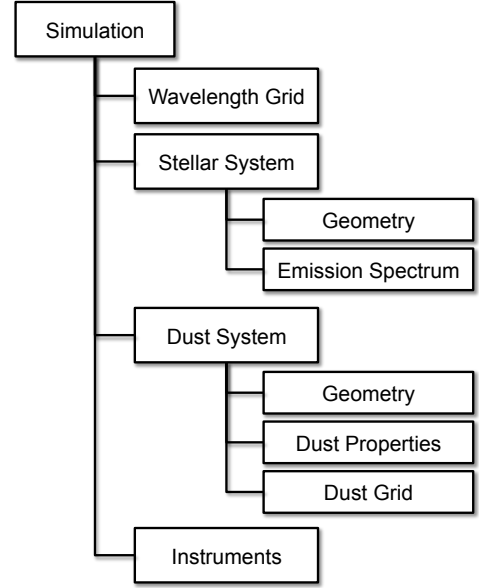


Figure 1: A schematic representation of the items to be configured for a particular SKIRT simulation.

components, and offers a variety of simulated instruments for measuring the radiation field from any angle. The code operates efficiently and is parallelized on shared memory systems.

SKIRT is a console application and is completely written in C++. It can easily be deployed on any Unix system, including for example Ubuntu and Mac OS X. The code has no compile-time options; all built-in components and capabilities are configured at run-time. The first-time user would use the interactive query and answer mechanism (in a terminal window) to configure a particular simulation. The smart mechanism guides the user through all possible options, narrowing down the possibilities based on earlier choices. The complete configuration for the simulation is then saved as a SKIRT parameter file in XML (eXtensible Markup Language) format, which can be easily viewed and adjusted in a regular text editor, even by an occasional user.

2.2. Configuring a simulation

Figure 1 illustrates the structure of a SKIRT simulation. Each of the building blocks offers several alternatives and options that can be configured in the parameter file. At the top level, for example, SKIRT supports two simulation types: oligochromatic and panchromatic. An *oligochromatic* simulation operates at just one or a small number of distinct wavelengths. It handles absorption and scattering by dust grains, but it doesn’t support thermal dust emission. There is no way to compute the dust temperature without integrating the absorbed radiation energy over an appropriate wavelength range. This basic simulation mode is appropriate for studying optical wavelengths, since the dust emission is negligible there. A *panchromatic* simulation operates over a broad range of wavelengths. These simulations can handle thermal dust emission as well as absorption and scattering, and thus many more options need to be configured.

The wavelength grid for an oligochromatic simulation is simply a short list of distinct wavelengths. A panchromatic simulation employs a grid over a range that typically extends from UV to millimeter wavelengths. SKIRT offers a plain logarithmic wavelength grid and a nested logarithmic wavelength grid, providing a higher resolution in some subset of the range. The user can specify the wavelength range and the number of grid points. Alternatively SKIRT can read a custom wavelength grid from a text data file that lists the grid points.

The spatial distribution of radiation sources and dust is obviously an important part of the simulation model. For this purpose SKIRT offers a number of predefined geometries; the most important ones are listed in Table 1(a). Each geometry defines a spatial density distribution, which can be used for radiation sources as well as dust components. Choices include a point-like source and various theoretical models for distributed densities with spherical, cylindrical, or no symmetries. Decorator geometries adjust another geometry by shifting its center to an arbitrary location, deforming a spherical geometry into a spheroidal or triaxial distribution, or adding clumps in random locations. Other geometries can import a density distribution from a data file. Anisotropic radiation sources are supported as well. Multiple geometries can be combined in arbitrary ways, enabling the construction of complex models.

The stellar system describes the radiation sources in the simulation model. For each geometry, the configuration defines the emission spectrum and the luminosity. Table 1(b) lists the built-in spectral energy distributions (SEDs), including several well-known parameterized SED families, and the option to import an SED from file. The amount of radiation can be specified through the bolometric luminosity or through the spectral luminosity at the center of a standard wavelength band. SKIRT also includes specialized stellar systems to import a snapshot from a hydrodynamic simulation using smoothed particles (SPH) or an adaptive mesh (AMR). In this case, both the spatial distribution and the emission spectrum in each location are extracted from the input data, for example using a Bruzual-Charlot model based on stellar age and metallicity.

Similarly, the dust system describes the spatial distribution and the properties of the dust in the model. A dust system can have multiple components, each with its own geometry and dust characterization. The amount of dust in each component can be defined simply as a total mass, or by specifying the optical depth along a particular axis. The optical and chemical properties of the dust in each component can be configured in great detail, as described in Sect. 2.3. Again, there are specialized dust systems to import a snapshot from a hydrodynamic simulation (SPH or AMR). The spatial distribution of the dust is now calculated from the gas density in the input data, assuming that the amount of dust is proportional to the metal fraction in the gas, except in areas where the gas is too hot to form dust.

The dust system also configures the dust grid, i.e. the computational structure that is used to discretize the spatial domain under study. The grid partitions the spatial domain in individual dust cells, and all physical variables (dust density, optical properties, radiation field, dust temperature) are considered to be constant in each dust cell. During the radiative transfer simu-

lation, photon packages propagate through the grid and interact with particular cells according to randomly generated events. Since memory requirements and computation time rapidly increase with the number of dust cells, a good grid has smaller cells in areas that require a higher resolution, and larger cells elsewhere. Table 1(d) lists the dust grids built into SKIRT. The spherical and cylindrical grids are perfect for simple models with the corresponding symmetries. Linear grids have equidistant grid points; logarithmic and power-law grids place (much) smaller bins in the central areas of the model. Most state-of-the-art simulations use 3D models, however, and SKIRT offers a choice of smart structured and unstructured grids to help optimize accuracy and performance. This is an active field of study in our research group, as described in Sect. 2.4.

Finally, the configuration is completed with a number of synthetic instruments, which collect and write down information about the simulated radiation received at some specified viewpoint. The *SED* instrument outputs the spectral energy distribution of the received flux as a text file that can easily be plotted. The *frame* instrument collects a complete 3D data cube (a rectangular frame of flux samples at each simulated wavelength) and outputs the result as a FITS file (Flexible Image Transport System), enabling the use of the standard visualization and data manipulation tools. The instruments can treat the radiation differently depending on its source; for example direct radiation, scattered radiation and dust emission can be recorded separately. The default instruments assume that the distance to the model is very large, so that they can use parallel projection. The perspective instrument, however, can be placed anywhere, even inside the model. It is mostly used to create animations by specifying an instrument per movie frame, with slightly varying position and/or angles.

2.3. Dust properties

The dust system in a SKIRT simulation can hold multiple dust components, each with their own spatial distribution and their specific dust characterization. The dust properties applying to a particular dust component are bundled in a building block called a *dust mix*. SKIRT offers several options to configure a dust mix, ranging from very simple to quite involved. Table 1(c) lists some of the choices. Each of the turn-key dust mixes implements a particular dust model described in the literature, usually including some specific combination of silicate grains, graphite grains, and polycyclic aromatic hydrocarbon (PAH) molecules, with properties listed in data files and/or approximated by formulae. These dust mixes can be configured simply by supplying their name.

Alternatively, the user can configure a custom dust mix from basic building blocks, as illustrated in Fig. 2. A configurable dust mix holds a distinct dust population for each type of grain material in the mix. For each type of grain material, the dust population specifies the optical and calorimetric material properties and a grain size distribution function. Optical dust properties include the scattering and absorption coefficients $\kappa^{\text{sca}}(\lambda, a)$ and $\kappa^{\text{abs}}(\lambda, a)$, and the asymmetry parameter $g(\lambda, a)$ determining the scattering phase function $\Phi_{\lambda,a}(\mathbf{k}, \mathbf{k}')$, for a range of wavelengths λ and a range of grain sizes a . Calorimetric proper-

Table 1: An overview of built-in components that can be used for defining (a) the spatial distribution of radiation sources and dust components, (b) the spectral energy distribution of radiation sources, (c) the properties of the dust mixture, and (d) the spatial discretization of the dust medium.

(a) Geometries	<i>Spherically symmetric</i>	
	PointGeometry	single point
	PlummerGeometry	classical Plummer sphere (Plummer 1911; Dejonghe 1987)
	SersicGeometry	spherical model with a Sérsic surface brightness profile (Sérsic 1963; Ciotti and Bertin 1999)
	EinastoGeometry	spherical model with an Einasto density profile (Einasto 1965; Retana-Montenegro et al. 2012)
	GammaGeometry	spherical model with a gamma density profile (Dehnen 1993; Tremaine et al. 1994)
	ShellGeometry	spherical shell where the density behaves as a power law between an inner and an outer radius
	<i>Axisymmetric</i>	
	ExpDiskGeometry	optionally truncated exponential profile in both radial and vertical directions (van der Kruit 1986)
	RingGeometry	ring with gaussian profile in the radial direction and exponential fall-off in the vertical direction
	TorusGeometry	torus with radial power-law profile within opening angle (Stalevski et al. 2012; Granato and Danese 1994)
	GaussianGeometry	model with gaussian distribution in the radial and the vertical direction
	MGEGeometry	geometry defined by a Multi-Gaussian Expansion (Emsellem et al. 1994; Cappellari 2002)
	<i>No symmetries</i>	
	ExpDiskSpiralArmsG...	double-exponential profile with a spiral arm perturbation (Misiriotis et al. 2000)
	AdaptiveMeshGeometry	density distribution defined over an adaptive mesh grid, imported from a data file
	VoronoiGeometry	density distribution defined over a Voronoi tessellation, imported from a data file
	<i>Decorators</i>	
	OffsetGeometry	applies an arbitrary offset to any other geometry
	ClumpyGeometry	replaces a portion of the mass in any geometry by randomly placed clumps
	SpheroidalGeometry	transforms any geometry to a spheroidal counterpart
	TriaxialGeometry	transforms any geometry to a triaxial counterpart
	<i>Anisotropic</i>	
	NetzerGeometry	point source with the anisotropic radiation profile of an accretion disk (Netzer 1987)
(b) SEDs	<i>Simple</i>	
	BlackBodySED	classical black body spectrum for a given temperature
	PegaseSED	SED templates for elliptical, lenticular and spiral galaxies (Fioc and Rocca-Volmerange 1997)
	QuasarSED	SED template for a quasar (Schartmann et al. 2005)
	StarburstSED	SED templates for a starbursting stellar population with given metallicity (Leitherer et al. 1999)
	SunSED	the solar spectrum
	FileSED	arbitrary spectrum imported from a data file
	<i>Families</i>	
	BruzualCharlotSED	stellar population SEDs parameterized on metallicity and age (Bruzual and Charlot 2003)
	MarastonSED	stellar population SEDs parameterized on metallicity and age (Maraston 1998)
	KuruczSED	stellar SEDs parameterized on metallicity, effective temperature and surface gravity (Kurucz 1993)
	MappingsSED	starbursting region SEDs par. on metallicity, compactness, pressure and covering factor (Groves et al. 2008)
(c) Dust Mixes	<i>Turn-key dust mixes</i>	
	DraineLiDustMix	mixture of graphite, silicate and PAH grains (Draine and Li 2007)
	MRNDustMix	mixture of graphite and silicate grains (Mathis et al. 1977; Weingartner and Draine 2001)
	WeingartnerDustMix	mixture of graphite, silicate and PAH grains (Weingartner and Draine 2001)
	ZubkoDustMix	mixture of graphite, silicate and PAH grains (Zubko et al. 2004)
	<i>Custom dust mixes</i>	
	ConfigurableDustMix	custom-configured dust mix given a list of grain compositions and grain size distributions
	<i>Grain compositions</i>	
	DraineGrainComp	optical and calorimetric properties for graphite, silicate and PAH grains (Draine and Li 2007)
	DustEmGrainComp	any of the dust grain properties provided with the DustEM code (Compiègne et al. 2011)
	ForsteriteGrainComp	Forsterite crystalline silicate grain properties (Fabian et al. 2001; Min et al. 2005; Suto et al. 2006)
	EnstatiteGrainComp	Enstatite crystalline silicate grain properties (Jaeger et al. 1998; Min et al. 2005)
	<i>Grain size distributions</i>	
	PowerLawGrainSize	modified power-law grain size distribution with a form inspired by Compiègne et al. (2011)
	LogNormalGrainSize	modified log-normal grain size distribution with a form inspired by Compiègne et al. (2011)
(d) Dust Grids	<i>Spherically symmetric</i>	
	LinSpheDustGrid	spherical grid with regularly spaced cells (radii of cell boundaries are equidistant)
	LogSpheDustGrid	spherical grid with logarithmically spaced cells (central cells have smaller widths)
	PowSpheDustGrid	spherical grid with cells spaced according to a power-law (central cells have smaller widths)
	<i>Axisymmetric</i>	
	LinAxDustGrid	cylindrical grid with regularly spaced cells in both radial and vertical directions
	LogLinAxDustGrid	cylindrical grid with logarithmically spaced cells radially, and regularly spaced cells vertically
	PowAxDustGrid	cylindrical grid where cells are spaced according to a power-law in both directions
	LogPowAxDustGrid	cylindrical grid with logarithmically spaced cells radially, and power-law spaced cells vertically
	<i>Cuboidal</i>	
	LinCubDustGrid	grid with regularly spaced cuboidal cells in the three dimensions
	PowCubDustGrid	grid with cuboidal cells spaced according to a power-law in the three dimensions
	OctTreeDustGrid	octree grid that recursively subdivides cuboidal nodes into eight sub-nodes (Saftly et al. 2013)
	BinTreeDustGrid	k-d tree grid that recursively subdivides cuboidal nodes into two sub-nodes (Saftly et al. 2014)
	<i>Unstructured</i>	
	VoronoiDustGrid	unstructured dust grid based on a Voronoi tessellation of 3D space (Camps et al. 2013)

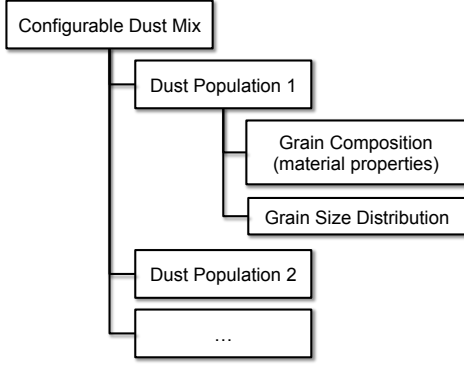


Figure 2: A schematic representation of a dust mix containing multiple dust populations. Each population describes a particular type of grain.

ties include the heat capacity $C(T)$ or equivalently the internal energy $U(T)$ of the dust grain material at a range of temperatures T , and the bulk mass density ρ_{bulk} of the material. Several sets of standard material properties and often-used size distributions are built-in to SKIRT, as illustrated in Table 1(c), and new choices can be easily added.

Configurable dust mixes allow users to experiment with new dust models, or to specify a different spatial distribution for a particular dust population (by splitting it off into a separate dust mix corresponding to a dust component with its own geometry).

2.4. Dust grids

The construction of a proper dust grid is a key aspect of a radiative transfer simulation. Many astrophysical models feature small structures, such as dust clumps or star forming regions, which require a lot of cells to resolve properly. To minimize memory requirements and computation time, the grid should be adapted to the spatial structure of the model. Therefore, in addition to fixed grids similar to the one shown in the top panel of Fig. 3, SKIRT offers several types of adaptive grids, including the k -d tree and the Voronoi grid shown in middle and bottom panels of the same figure.

Starting from a cuboidal root cell that spans the complete spatial domain, a typical adaptive mesh refinement (AMR) scheme recursively subdivides each cell into $a \times b \times c$ cuboidal subcells until sufficient resolution has been reached in each region. In the special case where $a = b = c = 2$, each cell is subdivided into eight subcells and the data structure is called an octree. The octree implementation in SKIRT was optimized in the context of radiative transfer as reported in Saftly et al. (2013).

A k -d tree (k -dimensional tree) is a space-partitioning data structure where each cell is recursively split into just two subcells along a particular hyperplane. In 3D space, i.e. with $k = 3$, a k -d tree is similar to an octree. In fact, any octree of depth n has an equivalent k -d tree of depth $3n$. For each octree level, the k -d tree uses three consecutive levels with mutually orthogonal dividing planes. However, while an octree forces all eight subcells to be created at the same time, a k -d tree allows more fine-grained control over which of the two initial subcells are subdivided further. As reported in Saftly et al. (2014), this property gives k -d tree grids a relevant advantage over octree grids

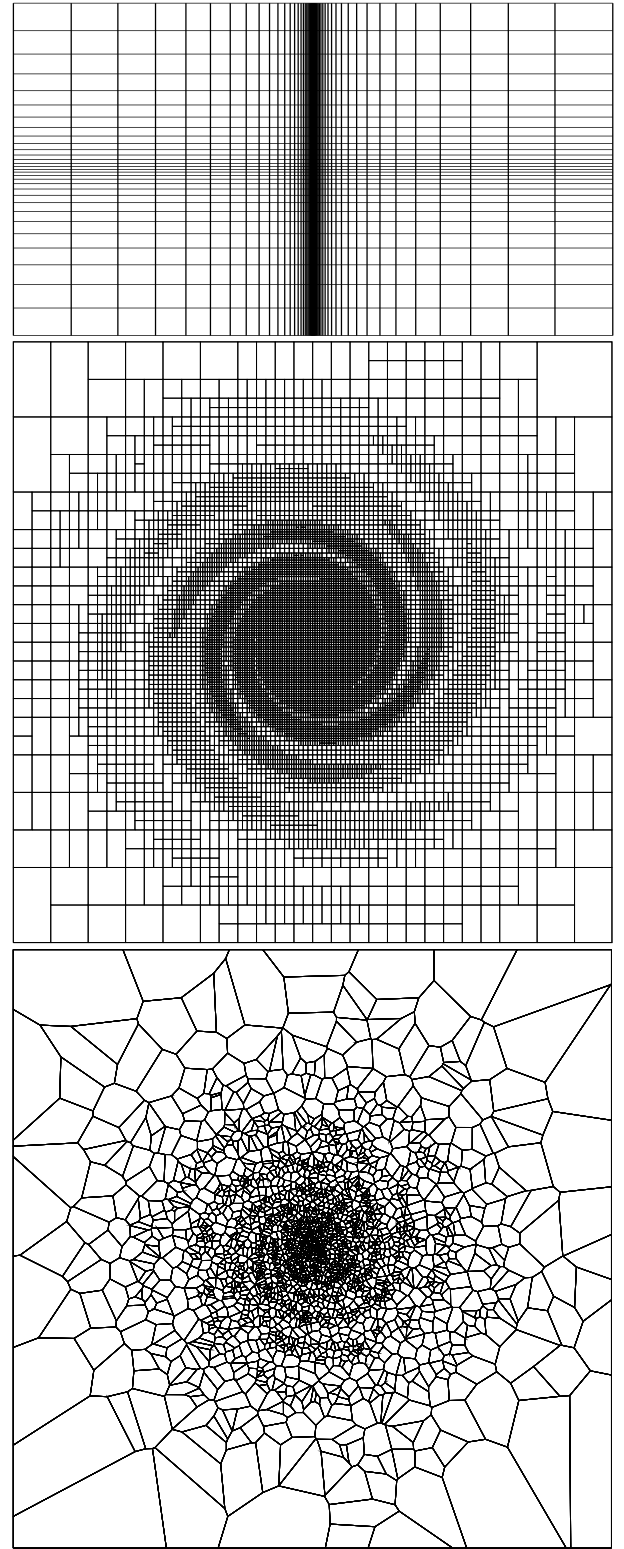


Figure 3: A planar cut through three of SKIRT's dust grids. *Top*: a fixed grid with cell sizes distributed logarithmically on the horizontal axis and according to a power-law on the vertical axis. *Center*: a cuboidal k -d tree grid with cell sizes that are adjusted to the dust density distribution in a simple spiral galaxy model. *Bottom*: an unstructured Voronoi grid where the generating sites are placed randomly following the dust density distribution of the same spiral galaxy model. Note that a planar cut through a 3D Voronoi tessellation is usually *not* a 2D Voronoi tessellation.

in the context of radiative transfer. The central panel of Fig. 3 shows a cut through a k -d tree grid with cell sizes that are adjusted to the dust density distribution in a simple spiral galaxy model.

Adaptive grids with cuboidal cells have become popular mainly because of their relative ease of implementation. But there is no a priori reason to assume that the cuboidal cell form is optimal. To the contrary, the strict coordinate-plane alignment of cell boundaries makes it hard to represent steep gradients in arbitrary directions, raising the number of cells needed to properly resolve clumpy features. One could consider constructing a grid using polyhedra instead of cuboids, but in general this seems a daunting task. Fortunately George Voronoi (Voronoi 1908) provided a specific way of partitioning 3D space into convex polyhedra. The mathematical properties of a Voronoi tessellation greatly facilitate implementation of an unstructured Voronoi grid in the context of radiative transfer, as described in Camps et al. (2013). Further research should determine whether Voronoi grids can indeed resolve astrophysical structures using less cells than cuboidal adaptive grids. The bottom panel of Fig. 3 shows a cut through a 3D Voronoi grid. Cells are placed randomly according to the dust density distribution of a simple spiral galaxy model.

2.5. User interface

The complete configuration for a particular SKIRT simulation is stored in a single parameter file, called a *ski file* (pronounced "skee file"). In view of the many features, options and interdependencies described in the previous sections, the contents of a ski file can become quite complex. To deal with this complexity, we opted for a file format based on XML (eXtensible Markup Language). This format has several advantages. XML elements can be nested to create hierarchies of features and options that reflect the natural makeup of the simulation's configuration. XML is stored as plain text, so it can be easily viewed and adjusted in a regular text editor, even by an occasional user; the human-readable XML tags make the format self-explanatory to a large degree. And finally, existing ski files remain compatible when new features are added (with appropriate defaults), or can be automatically upgraded when the structure changes in an incompatible way.

To perform a simulation, the user starts the code in a terminal window, supplying the name of the relevant ski file on the command line. The code runs fully unattended and all results are written to output files. A small number of command line options allow overriding some defaults in the run-time environment, such as the number of parallel threads or the location of input and output files. The makeup of the simulation itself is fully defined in the ski file. When SKIRT is started without any command line arguments, it enters an interactive query and answer mode that guides the user through the process of creating a new ski file.

To illustrate how this works, we configure a SKIRT simulation for a simple spiral galaxy model, with an instrument that produces the edge-on view shown in Fig. 4. The query and answer session in the terminal window is illustrated in Fig. 5. The

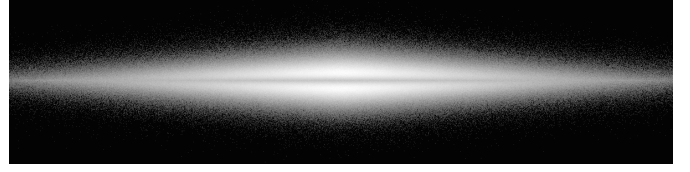


Figure 4: The output of a SKIRT simulation for a very simple spiral galaxy model with an instrument that registers the total flux of an edge-on view. The SKIRT parameter file for this simulation is shown in Fig. 6.

resulting ski file is shown in Fig. 6, and a pretty-printed version is shown in Fig. 7.

When configuring a particular type of simulation for the first time, the query and answer mechanism guides the user through all possible options, narrowing down the possibilities based on earlier choices. This is similar to the concept of a *wizard* in graphical user interfaces. Subsequently, the user can easily adjust the constructed ski file in a text editor; a slightly more experienced user can copy and paste building blocks between different ski files. For each simulation performed, SKIRT produces a \LaTeX file describing the contents of the input ski file in a human-readable format that can be used for documentation purposes.

To further facilitate the configuration process, physical quantities such as distances, sizes or masses can be specified in units selected by the user. The default unit system for a simulation's input and output is specified early on in the ski file (e.g. extragalactic units on line 5 of Fig. 6), and individual parameter values can be specified with a units string that overrides the default. For example, a scale length of 6600 pc could be specified as "6600 pc", "6.6 kpc", or approximately "2e20 m".

3. Architecture

The latest version of SKIRT was re-architected with the following major design goals in mind:

- *Structured parameter file*: use a self-documenting ski file format that supports the complex configuration needs described above in a user-friendly manner.
- *Single point of definition*: define all information relating to a new feature only once and in the same place, including the code, the human-readable text strings used in the query and answer session, and the tags in the ski file.
- *Data-driven user interface*: conduct the query and answer session and handle the ski file based solely on these data definitions, so that the user interface adjusts automatically as new features are added.
- *Shared-memory parallelization*: make all code reentrant by eliminating the use of global variables; protect the remaining global resources or writable shared data with appropriate locking mechanisms.
- *Modularity*: minimize dependencies among different areas of the code by providing appropriate interfaces and data encapsulation.

```

1 $ skirt
2 ? Enter the name of the ski file to be created: spiralgalaxy
3 Possible choices for the simulation:
4   1. An oligochromatic Monte Carlo simulation
5   2. A panchromatic Monte Carlo simulation
6 ? Enter one of these numbers [1,2] (1): 1
7 Possible choices for the units system:
8   1. SI units
9   2. Stellar units (length in AU, distance in pc)
10  3. Extragalactic units (length in pc, distance in Mpc)
11 ? Enter one of these numbers [1,3] (3):
12 ? Enter the number of photon packages per wavelength [0,2e13] (1e6): 1e7
13 Possible choices for the wavelength grid:
14   1. A list of one or more distinct wavelengths
15 Automatically selected the only choice: 1
16 ? Enter the wavelengths [0.0001 micron,1e6 micron]: 0.55
17 Possible choices for the stellar system:
18   1. A stellar system composed of various stellar components
19   2. A stellar system derived from an SPH output file
20 ...
21 ? Enter one of these numbers [1,4] (1): 1
22 ...
23 Possible choices for the geometry of the dust component:
24   1. A point source geometry
25   2. A Plummer geometry
26   ...
27   9. An exponential disk geometry
28   ...
29 ? Enter one of these numbers [1,26] (9): 9
30 ? Enter the radial scale length [0 pc,∞ pc]: 6600
31 ? Enter the axial scale height [0 pc,∞ pc]: 250
32 ? Enter the radial truncation length (zero means no truncation) [0 pc,∞ pc] (0 pc):
33 ? Enter the axial truncation height (zero means no truncation) [0 pc,∞ pc] (0 pc):
34 Possible choices for the dust mixture of the dust component:
35   1. A Draine & Li (2007) dust mix
36   ...
37 ? Enter one of these numbers [1,11] (2): 1
38 Possible choices for the type of normalization for the dust component:
39   1. Normalization by defining the total dust mass
40   2. Normalization by defining the edge-on optical depth at some wavelength
41   ...
42 ? Enter one of these numbers [1,7] (1): 1
43 ? Enter the total dust mass of the dust component [0 Msun,∞ Msun]: 4e7
44 Possible choices for item #2 in the dust components list:
45   1. A dust component
46 ? Enter one of these numbers or zero to terminate the list [0,1] (1): 0
47 Possible choices for the dust grid structure:
48   1. A cylindrical grid structure with a linear distribution
49   ...
50   4. A cylindrical grid structure with a radial logarithmic and axial power-law distribution
51   ...
52   13. A Voronoi dust grid structure
53 ? Enter one of these numbers [1,13] (10): 4
54 ? Enter the inner radius in the radial direction [0 pc,∞ pc]: 250
55 ? Enter the outer radius in the radial direction [0 pc,∞ pc]: 25000
56 ? Enter the number of radial grid points [5,100000] (250): 101
57 ? Enter the outer radius in the axial direction [0 pc,∞ pc]: 7000
58 ? Enter the ratio of the inner- and outermost bin widths in the axial direction [0,1e4] (50): 50
59 ? Enter the number of axial grid points [5,100000] (250): 101
60 Successfully created ski file: spiralgalaxy.ski
61 $

```

Figure 5: A partial transcript of the query and answer terminal session to configure a SKIRT simulation for a simple spiral galaxy model. The smart mechanism guides the user through all possible options, narrowing down the possibilities based on earlier choices. For example, on line 15 there is only one choice for the wavelength grid because on line 6 the user selected an oligochromatic simulation. Also the dust grid choices on lines 47-52 are limited to 2D and 3D grids (omitting 1D grids) since the geometry selected on line 29 is axisymmetric. Furthermore the options for the geometry in lines 30-33 and for the dust grid in lines 54-59 are tailored to the selected type of geometry/dust grid.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <skirt-simulation-hierarchy type="MonteCarloSimulation" format="6.1">
3   <OligoMonteCarloSimulation packages="1e7">
4     <units type="Units">
5       <ExtragalacticUnits/>
6     </units>
7     <wavelengthGrid type="OligoWavelengthGrid">
8       <OligoWavelengthGrid wavelengths="0.55 micron"/>
9     </wavelengthGrid>
10    <stellarSystem type="StellarSystem">
11      <CompStellarSystem>
12        <components type="StellarComp">
13          <OligoStellarComp luminosities="1e11">
14            <geometry type="Geometry">
15              <ExpDiskGeometry radialScale="4400 pc" axialScale="500 pc"
16                radialTrunc="0 pc" axialTrunc="0 pc"/>
17            </geometry>
18          </OligoStellarComp>
19        </components>
20      </CompStellarSystem>
21    </stellarSystem>
22    <dustSystem type="OligoDustSystem">
23      <OligoDustSystem>
24        <dustDistribution type="DustDistribution">
25          <CompDustDistribution>
26            <components type="DustComp">
27              <DustComp>
28                <geometry type="Geometry">
29                  <ExpDiskGeometry radialScale="6600 pc" axialScale="250 pc"
30                    radialTrunc="0 pc" axialTrunc="0 pc"/>
31                </geometry>
32                <mix type="DustMix">
33                  <DraineLiDustMix/>
34                </mix>
35                <normalization type="DustCompNormalization">
36                  <DustMassDustCompNormalization dustMass="4e7 Msun"/>
37                </normalization>
38              </DustComp>
39            </components>
40          </CompDustDistribution>
41        </dustDistribution>
42        <dustGridStructure type="DustGridStructure">
43          <LogPowAxDustGridStructure
44            radialInnerExtent="250 pc" radialOuterExtent="25000 pc" radialPoints="101"
45            axialExtent="7000 pc" axialRatio="50" axialPoints="101"/>
46          </dustGridStructure>
47        </OligoDustSystem>
48      </dustSystem>
49    <instrumentSystem type="InstrumentSystem">
50      <InstrumentSystem>
51        <instruments type="Instrument">
52          <FrameInstrument instrumentName="xz" distance="10 Mpc"
53            inclination="90 deg" azimuth="-90 deg" positionAngle="0 deg"
54            pixelsX="1200" extentX="28000 pc" pixelsY="300" extentY="7000 pc"/>
55        </instruments>
56      </InstrumentSystem>
57    </instrumentSystem>
58  </OligoMonteCarloSimulation>
59 </skirt-simulation-hierarchy>

```

Figure 6: The ski file (SKIRT parameter file) configured during the query and answer session shown in Fig. 5. While it would be hard for a human to create this file from scratch, it is surprisingly readable because of the self-explanatory tag names. For example, it is easy even for a casual user to adjust the scale height of the dust lane on line 29 or to add an extra instrument by copying lines 52-54 and modifying the inclination angle of the second instrument.

```

1 SKIRT parameter overview: spiralgalaxy
2
3 An oligochromatic Monte Carlo simulation
4 . The random number generator: the default random generator
5 . . The seed for the random generator: 4357
6 . The units system: extragalactic units (length in pc, distance in Mpc)
7 . The instrument system: an instrument system
8 . . Item #1 in the instruments list: a basic instrument that outputs the total flux in every pixel as a data cube
9 . . . The name for this instrument: xz
10 . . . The distance to the system: 10 Mpc
11 . . . The inclination angle  $\theta$  of the detector:  $90^\circ$ 
12 . . . The azimuth angle  $\varphi$  of the detector:  $-90^\circ$ 
13 . . . The position angle  $\omega$  of the detector:  $0^\circ$ 
14 . . . The number of pixels in the horizontal direction: 1200
15 . . . The maximal horizontal extent: 28000 pc
16 . . . The number of pixels in the vertical direction: 300
17 . . . The maximal vertical extent: 7000 pc
18 . The number of photon ( $\gamma$ ) packages per wavelength:  $1 \times 10^7$ 
19 . The wavelength grid: a list of one or more distinct wavelengths
20 . . The wavelengths ( $\lambda$ ):  $0.55 \mu\text{m}$ 
21 . The stellar system: a stellar system composed of various stellar components
22 . . Item #1 in the stellar components list: a stellar component in an oligochromatic simulation
23 . . . The geometry of the stellar distribution: an exponential disk geometry
24 . . . . The radial scale length: 4400 pc
25 . . . . The axial scale height: 500 pc
26 . . . . The radial truncation length (zero means no truncation): 0 pc
27 . . . . The axial truncation height (zero means no truncation): 0 pc
28 . . . The luminosities, one for each wavelength, in solar units:  $1 \times 10^{11}$ 
29 . The dust system: a dust system for use with oligochromatic simulations
30 . . The dust distribution: a dust distribution composed of various dust components
31 . . . Item #1 in the dust components list: a dust component
32 . . . . The geometry of the dust component: an exponential disk geometry
33 . . . . . The radial scale length: 6600 pc
34 . . . . . The axial scale height: 250 pc
35 . . . . . The radial truncation length (zero means no truncation): 0 pc
36 . . . . . The axial truncation height (zero means no truncation): 0 pc
37 . . . . The dust mixture of the dust component: a Draine & Li (2007) dust mix
38 . . . . . Output a data file with the optical properties of the dust mix: yes
39 . . . . . Output a data file with the mean optical properties of the dust mix: yes
40 . . . . The type of normalization for the dust component: normalization by defining the total dust mass
41 . . . . . The total dust mass of the dust component:  $4 \times 10^7 M_\odot$ 
42 . . The dust grid structure: a cylindrical grid structure with a radial logarithmic and axial power-law distribution
43 . . . Output data files for plotting the structure of the grid: yes
44 . . . The inner radius in the radial direction: 250 pc
45 . . . The outer radius in the radial direction: 25000 pc
46 . . . The number of radial grid points: 101
47 . . . The outer radius in the axial direction: 7000 pc
48 . . . The ratio of the inner- and outermost bin widths in the axial direction: 50
49 . . . The number of axial grid points: 101
50 . . The number of random density samples for determining cell mass: 100
51 . . Output a data file with convergence checks on the dust system: yes
52 . . Output FITS files displaying the dust density distribution: yes
53 . . Calculate and output quality metrics for the dust grid: no
54 . . Output a data file with relevant properties for all dust cells: no
55 . . Output statistics on the number of cells crossed per path: no

```

Figure 7: A pretty-printed version of the ski file (SKIRT parameter file) shown in Fig. 6. SKIRT produces a \LaTeX file describing the configuration in this way for each simulation performed. The description includes any default values that were omitted from the ski file; see for example lines 50-55.

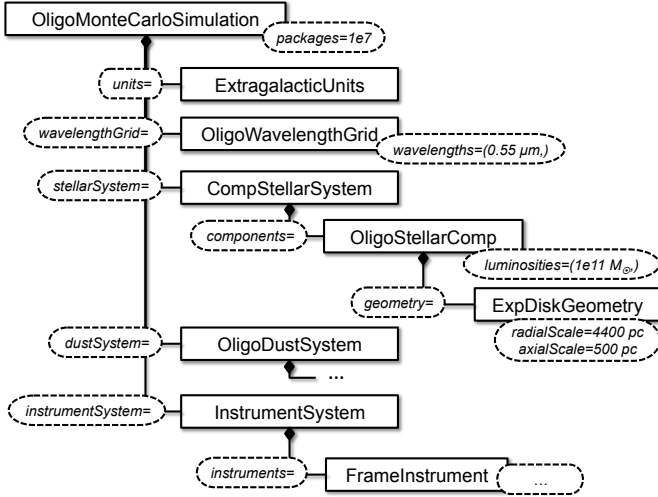


Figure 8: The run-time object hierarchy that would be constructed for the ski file shown in Fig. 6. A solid rectangle represents a simulation item of the specified type; a dashed oval indicates the name and value of a (plain or composite) property. Connections starting with a diamond indicate aggregation. Each simulation item instance and each property in this hierarchy maps directly to an XML element or attribute in the ski file with the same name.

In this section we describe the overall architecture of the code and we point out how it achieves these design goals. SKIRT is written in C++ using object-oriented design principles. Specifically we use several of the design patterns originally described by Gamma et al. (1994) in their classic work, including for example the *Composite*, *Builder*, *Visitor*, and *Decorator* patterns.

3.1. Simulation items

The core of the SKIRT code is obviously about performing radiative transfer simulations. A complete SKIRT simulation is represented at run-time as a hierarchy of objects called *simulation items*, similar to the structure illustrated in Fig. 1. This object hierarchy represents the configuration of the simulation (in its structural makeup and in some data members), offers the functionality to perform the simulation (through its member functions), and provides space for any intermediate and resulting data structures (in its data members). Multiple simulation object hierarchies can co-exist and are fully independent of each other.

The object hierarchy for a particular simulation closely mimics the structure of the corresponding ski file. For example, Fig. 8 shows the hierarchy that would be constructed for the ski file listed in Fig. 6. A solid rectangle represents a simulation item of the specified type; a dashed oval indicates the name and value of an property. Plain properties hold a single value (or a list of values); composite properties link other simulation items into the hierarchy. The simulation items and attributes in this hierarchy map directly to an XML element or attribute in the ski file with the same name. This correspondence plays an important role in the automation of the user interface, as we will discuss in Sect. 3.5.

Each simulation item is an instance of a class that directly or indirectly derives from the `SimulationItem` base class. The

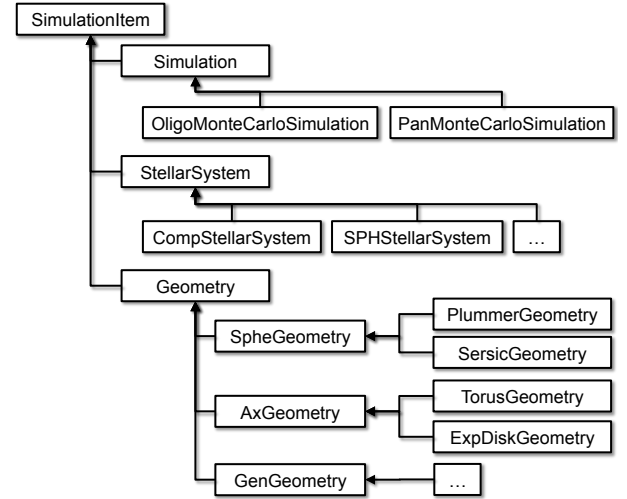


Figure 9: A small portion of the more than 150 simulation item classes in the compile-time inheritance hierarchy. A solid rectangle represents a simulation item class with the specified name. Connections starting with an inverted arrow indicate inheritance.

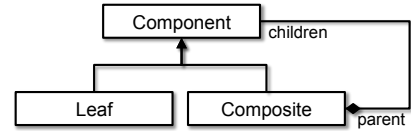


Figure 10: The *Composite* design pattern (Gamma et al. 1994). Connections starting with an inverted arrow indicate inheritance; connections starting with a diamond indicate aggregation. This pattern describes an aggregation of objects that all have the same base type.

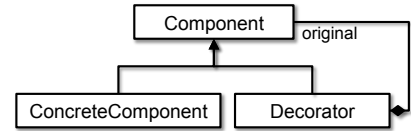


Figure 11: The *Decorator* design pattern (Gamma et al. 1994). Connections starting with an inverted arrow indicate inheritance; connections starting with a diamond indicate aggregation. This pattern describes a convenient way to adjust or *decorate* the behavior of another object of the same base type.

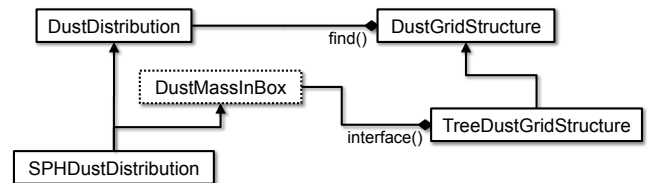


Figure 12: A specialty interface (dotted rectangle) connects two specific simulation items at run time, optimizing performance without creating undesirable dependencies in the respective base classes. Connections starting with an inverted arrow indicate inheritance; connections starting with a diamond indicate aggregation.

```

1 void SimulationItem::setup()
2 {
3     setupSelfBefore();
4     for (SimulationItem* child : children())
5     {
6         child->setup();
7     }
8     setupSelfAfter();
9 }

```

Figure 13: The implementation of the `SimulationItem::setup()` function (ignoring some implementation details).

simulation item classes form a compile-time inheritance hierarchy, a small portion of which is shown in Fig. 9. The run-time object hierarchy representing a simulation is thus an aggregation of objects of the same type, reflecting the *Composite* design pattern (Gamma et al. 1994) illustrated in Fig. 10. The *Component* role is played by the `SimulationItem` class, and the *Composite* role is assumed by any `SimulationItem` subclass that has one or more composite properties.

The use of the *Composite* pattern is fundamental to the implementation of the user interface discussed in Sect. 3.5, and it substantially facilitates reducing dependencies between portions of the code, as described in Sect. 3.2 and Sect. 3.3.

3.2. Simulation phases

A SKIRT simulation has three phases: *construction*, *setup* and *run*. In the *construction* phase, the code constructs the simulation item hierarchy corresponding to a particular ski file, initializing the values of all plain and composite properties, as described in Sect. 3.5. This process completes in a fraction of a second because it doesn't do much work. During the *setup* phase, each simulation item in the hierarchy gets a chance to perform further initialization, such as reading data from resource files or pre-computing frequently used information. This phase may require some processing power, for example, to set up a dust grid that is adapted to the specified dust distribution. Finally, the *run* phase performs the actual simulation and writes down the results. Usually this phase consumes the bulk of the computing resources, and it is fully parallelized.

The `SimulationItem` base class offers the `setup()` function; its implementation is shown in Fig. 13. The `children()` function used on line 4 returns a list of all simulation items held by any composite property of the current simulation item. The `setupSelfBefore()` and `setupSelfAfter()` functions are declared `virtual` in the `SimulationItem` base class and are overridden by subclasses that need to perform initialization during the *setup* phase. They are invoked respectively before and after any children of the simulation item have been set up.

The implementation of the `setup()` function follows the *Template Method* design pattern (Gamma et al. 1994) to delegate the actual initialization work to subclasses. To recursively invoke all simulation items in the hierarchy, it relies on the fact that all simulation item classes derive from the same class, which is ensured by the use of the *Composite* design pattern.

SKIRT requires that the root object of the simulation item hierarchy inherits from the `Simulation` class. This class offers

the `run()` function, which, not surprisingly, executes the *run* phase. Thus, after constructing the run-time hierarchy, the code simply invokes the `setup()` and `run()` functions on the hierarchy's root object to complete all phases of the simulation.

3.3. Reducing dependencies

Most simulation item classes are organized in groups with a common purpose, e.g. wavelength grids, geometries, or dust mixes. All classes in a particular group inherit from the same base class, i.e. `WavelengthGrid`, `Geometry`, or `DustMix`. The base class offers the common interface for all classes in the group towards classes outside of the group. This design principle avoids undesirable dependencies between classes in different groups, enhancing modularity.

Some information about a simulation's configuration is accessed from many different places, and thus must be readily available. To facilitate access to other simulation items in the same hierarchy, the `SimulationItem` class offers the `T* find<T>()` template function, where `T` stands for the name of any class that derives from `SimulationItem`. This template function searches the object hierarchy in which the receiving simulation item resides for a simulation item of the specified type `T`, and returns a pointer to the first such object found after dynamically casting it to the requested type. If the hierarchy does not contain an object of the specified type, the function throws an exception. The implementation of the `find<>()` template function again relies on the fact that all simulation item classes derive from the same class.

For example, every run-time simulation hierarchy includes an instance of a particular `WavelengthGrid` subclass, such as `OligoWavelengthGrid` or `NestedLogWavelengthGrid`. Any simulation item in the hierarchy can call `find<WavelengthGrid>()` to retrieve a pointer to the common wavelength grid interface; the caller does not know the specific sub-type of the returned object. Also, the caller has no need to know where the returned object resides, so this mechanism replaces application-wide global data (which gets in the way of parallelization) by simulation-wide available data.

While modularity is important, the generic and narrow interfaces between different areas of the code sometimes hide information that can be relevant for optimal cooperation between components. As a first example, the dust mass in each grid cell is usually estimated by probing the dust density distribution in a number of random locations uniformly distributed over the spatial extent of the cell. This generic mechanism works for any cell shape and for any type of density distribution. However, for certain cell shapes combined with certain types of density distribution, it might be orders of magnitude faster to directly calculate the mass in the cell. As a second example, sometimes we would like to build a dust grid based on particular locations (such as SPH particles) defined as part of the input dust density distribution, rather than based on the density distribution itself. However, the generic interface does not offer particle information because the concept is meaningless for most density distributions.

These features can be accomplished without breaking modularity by using a specialty interface that is known only to the

specific classes involved, and by providing a mechanism to dynamically connect the two players at run time. This is illustrated in Fig. 12 for the first example described above. The `DustMassInBox` interface declares pure virtual functions offering the relevant special capabilities. The `SPHDustDistribution` class inherits the interface and actually implements its functions. Finally, the `TreeDustGridStructure` simulation item recovers a pointer to the specialty interface as follows. First it uses the `find<>()` template function to retrieve the dust distribution object in the hierarchy; this could in fact happen in the `DustGridStructure` base class since the returned pointer is of the generic type `DustDistribution`. Then it invokes the `interface<>()` template function on the dust distribution object, which is in fact of type `SPHDustDistribution`, to return a pointer to the `DustMassInBox` interface implemented by that same object.

For this purpose, the `SimulationItem` base class provides the `T* interface<T>()`, where `T` stands for the name of the specialty interface to be recovered. In the example of Fig. 12, the function can be implemented with a simple dynamic cast. To support more complicated cases, the function also allows a simulation item to delegate the implementation of a specialty interface to a different object.

3.4. Reusing components

In a Monte Carlo radiative transfer code, the key function of a dust geometry (describing the distribution of the dusty medium) is to retrieve the dust density at a specified location in space. This function is called (quite often) during setup to build an appropriate dust grid and calculate the dust mass in each grid cell. On the other hand, the key function of a stellar geometry (describing the distribution of radiation sources) is to generate a random location in space, drawn from a probability distribution corresponding to the geometry’s density distribution. This function is called repeatedly during the simulation to determine the point of emission for a new photon package.

It might seem that the functionalities of the two geometry types are thus rather disjunct, but this is not the case. For example, we want to build a Voronoi dust grid using generating sites placed according to the dust density distribution. In this case, we need the key stellar geometry functionality (generating random points) in the dust geometry. Therefore the recent SKIRT version has unified geometry classes that offer both functions, as listed in Table 1(a).

This change prompted us to invest in a number of geometry classes that modify other geometry’s in interesting ways, following the *Decorator* design pattern (Gamma et al. 1994) shown in Fig. 11. An object in the *Decorator* role maintains a pointer to another object of the same base class, called the original object. The decorator implements the base class interface by calling corresponding functions in the original object, and returning the results after possible adjustment. In SKIRT, the `Geometry` class plays the *Component* role in this pattern, and any `Geometry` subclass can assume the *ConcreteComponent* role, i.e. the role of the original geometry being decorated. The `ClumpyGeometry` class is one of the classes that plays the *Decorator* role. It modifies the original geometry by replacing a

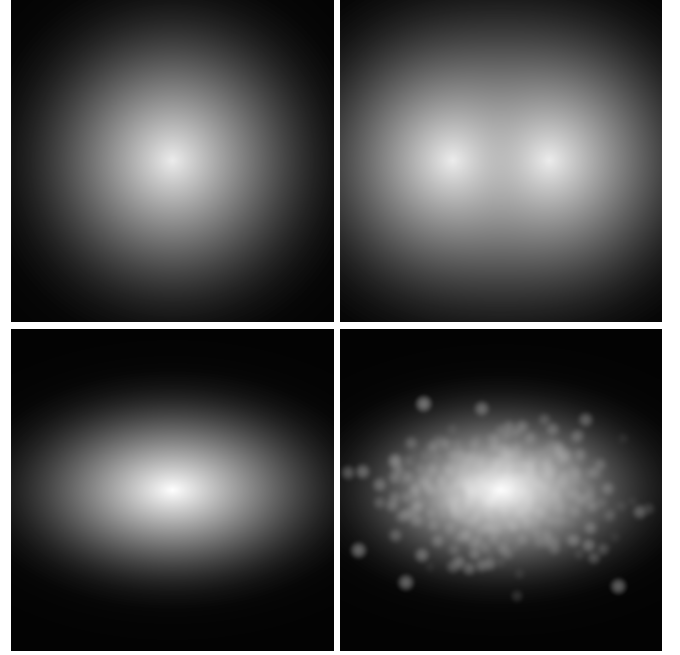


Figure 14: Planar cuts through four density distributions derived from the same underlying geometry using the *Decorator* design pattern: a plain, spherical Einasto profile with index 1 (*top left*); two superposed Einasto profiles, each shifted aside using an offset decorator (*top right*); the Einasto profile deformed into a spheroidal shape by a decorator (*bottom left*); and a clumpy, spheroidal Einasto distribution derived from the original profile by applying a chain of two decorators (*bottom right*).

fraction of its total mass allocation by randomly placed clumps. Other decorators relocate the original geometry’s center, or deform a spherical geometry into a spheroidal or triaxial distribution. Multiple decorators can be chained to achieve the combined effects, as illustrated in Fig. 14.

3.5. Automating the user interface

The latest version of SKIRT is based on the Qt development framework³, which includes a rich set of cross-platform C++ libraries and an integrated development environment (IDE) called Qt Creator. Although we don’t need its graphical user interface (GUI) capabilities, the Qt environment offers substantial benefits in other areas as well⁴. Specifically, the Qt environment provides run-time introspection of classes and their member functions, assuming the appropriate declarations were added in the code. The Qt mechanism is a lot more advanced than the standard C++ run-time type information (RTTI) system. For example, the Qt library offers functions to retrieve compile-time information such as the class inheritance hierarchy and the type of function arguments or return values. It is also possible to invoke a function by specifying the function name at run-time as a string, or to construct a new class instance in a similar manner.

SKIRT relies on the Qt introspection features to automatically construct a user interface from the C++ class declarations

³The Qt project: <http://qt-project.org>

⁴The new features in the recent C++11 language standard cover much of the functionality for which SKIRT uses Qt, with the exception of the introspection capabilities discussed in this section.

```

1  class ClumpyGeometry : public Geometry
2  {
3      Q_OBJECT
4      Q_CLASSINFO("Title", "a geometry that adds clumpiness to any geometry")
5
6      Q_CLASSINFO("Property", "geometry")
7      Q_CLASSINFO("Title", "the geometry to which clumpiness is added")
8
9      Q_CLASSINFO("Property", "clumpFraction")
10     Q_CLASSINFO("Title", "the fraction of the mass locked up in clumps")
11     Q_CLASSINFO("MinValue", "0")
12     Q_CLASSINFO("MaxValue", "1")
13
14     Q_CLASSINFO("Property", "clumpCount")
15     Q_CLASSINFO("Title", "the total number of clumps")
16     Q_CLASSINFO("MinValue", "1")
17
18     Q_CLASSINFO("Property", "clumpRadius")
19     Q_CLASSINFO("Title", "the scale radius of a single clump")
20     Q_CLASSINFO("Quantity", "length")
21     Q_CLASSINFO("MinValue", "0")
22
23     Q_CLASSINFO("Property", "cutoff")
24     Q_CLASSINFO("Title", "cut off clumps at the boundary of the underlying geometry")
25     Q_CLASSINFO("Default", "no")
26
27 public:
28     Q_INVOKABLE ClumpyGeometry();
29
30 protected:
31     void setupSelfBefore();
32     void setupSelfAfter();
33
34 public:
35     Q_INVOKABLE void setGeometry(Geometry* value);
36     Q_INVOKABLE Geometry* geometry() const;
37
38     Q_INVOKABLE void setClumpFraction(double value);
39     Q_INVOKABLE double clumpFraction() const;
40
41     Q_INVOKABLE void setClumpCount(int value);
42     Q_INVOKABLE int clumpCount() const;
43
44     Q_INVOKABLE void setClumpRadius(double value);
45     Q_INVOKABLE double clumpRadius() const;
46
47     Q_INVOKABLE void setCutoff(bool value);
48     Q_INVOKABLE bool cutoff() const;
49
50 public:
51     double density(Position bfr) const;
52     Position generatePosition() const;
53
54     ...
55 };

```

Figure 15: A typical simulation item class declaration. The keywords starting with `Q_` are provided by the Qt development environment, and serve to define the extra information needed to automatically build a user interface for the features offered by this class, as explained in Sect. 3.5. The setup functions declared on lines 31-32 are described in Sect. 3.2 and Fig. 13. The geometry-specific functions declared on lines 51-52 are described in Sect. 3.4.

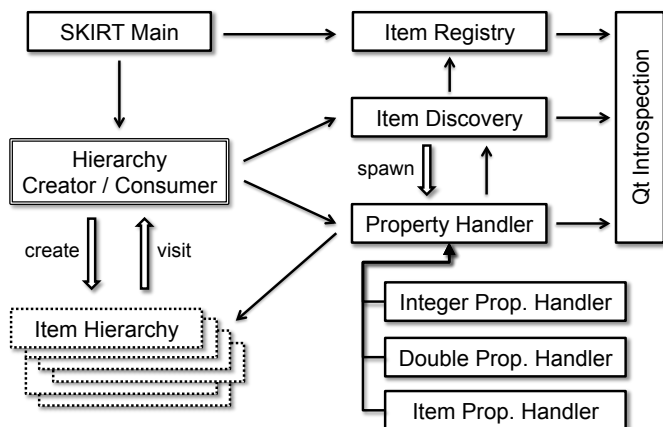


Figure 16: Schematic overview of the code that automatically builds the user interface from simulation item class declarations. In this diagram, the phrase *simulation item* has been shortened to *item*. A solid arrow indicates that the source module uses the target module. Connections starting with an inverted arrow indicate inheritance. The relationships are described in more detail in Sect. 3.5.

in the code. To enable this process, all `SimulationItem` subclass declarations must be augmented with some extra information, as illustrated in Fig. 15. The keywords starting with `Q_` are provided by the Qt development environment. The `Q_OBJECT` keyword on line 3 is required to enable the Qt introspection features for this class. The `Q_CLASSINFO` definitions on lines 4-25 associate an ordered list of key-value pairs with the compile-time class information; these strings can be retrieved at run-time through the Qt introspection system. The `Q_INVOKABLE` keyword on lines 28 and 35-48 enables Qt introspection for the constructor or member function declaration following the keyword.

In SKIRT, the `Q_CLASSINFO` key-value pairs are used to provide a human readable description for the class, and to define its configurable properties (i.e. the properties that can be specified in a ski file). The name of a property, e.g. "clumpFraction", must match the name of a getter function `clumpFraction()` and a setter function `setClumpFraction()`, declared with the keyword `Q_INVOKABLE`. The type of the configurable property is derived from the getter's return value (double in this case). Additional key-value pairs can specify options such as a default value, or the type of physical quantity represented by this property, which determines the units used or accepted in the user interface.

Figure 16 depicts the overall organization of the code that automatically builds the user interface from the compile-time data. The architecture is inspired by – but doesn't correspond exactly to – the *Builder* and *Visitor* design patterns (Gamma et al. 1994). The *hierarchy creator* object in the figure plays the *Builder* role, and the *hierarchy consumer* object plays the *Visitor* role.

Just after program startup, the *item registry* is initialized with a list of all simulation item classes. This ensures that all classes are actually linked into the code, and it provides the starting point for the *item discovery* module to implement queries about the simulation item classes. Functions offered by this module include for example `title(itemType)`, `descen-`

`dants(itemType)`, and `createPropertyHandlers(item)`. The latter function spawns a property handler of the appropriate type for each property of the specified simulation item.

A property handler combines a pointer to a particular simulation item object in the hierarchy, with knowledge about the compile-time attributes of one of the properties of the item. The handler can be used to get or set the property value directly from or into the target object, or to retrieve attributes such as its default value. The handler also knows how to convert a property value into a string for human consumption, and vice versa. There are handlers for various property types, including Boolean, integer, enumeration, floating point (with support for units), string, and pointer to simulation item.

To create a simulation item hierarchy from a ski file, the command line handler in the *SKIRT main* module enlists an `XmlHierarchyCreator` object. This object uses the XML tags in the ski file, which correspond to simulation class names and property names, to recursively construct the corresponding simulation items and set their property values. The code heavily relies on the *item discovery* module and the property handlers spawned by it.

Similarly, a `ConsoleHierarchyCreator` object is used to create a hierarchy from scratch by conducting a query and answer session. The top of the hierarchy must be occupied by an instance of the `Simulation` class, so the algorithm obtains a list of concrete `Simulation` subclasses from the *item discovery* module, and asks the user to make a choice. The question is formulated using the titles provided in each class declaration; see lines 4-6 in Fig. 5. The algorithm constructs an instance of the selected subclass, and then loops over all of its configurable properties, asking the appropriate question(s) for each property depending on its type. Boolean, numeric and string properties only need a single question; see lines 30-33 and 54-59 in Fig. 5. A property that points to another simulation item prompts a multiple choice question to select one of the available concrete subclasses that inherit the appropriate type, again obtained from the *item discovery* module; see lines 7-11 and 23-29 in Fig. 5. A new simulation item of the selected type is created (and linked into the hierarchy), and the same mechanism is recursively applied to the new object.

By selecting the desired type of simulation item at each level in the recursion, the user's responses drive the nature of subsequent questions in the session. While this is sufficient for most purposes, the discovery process implements a few extra mechanisms to support specific needs. For example, the list of available dust grids depends on the (lack of) symmetries in the geometries selected earlier; e.g. the user can't select a 1D or 2D grid for a 3D geometry. Also, it is possible to skip questions that are deemed irrelevant based on the response to a previous question in the same class. All of these mechanisms are fully data-driven from the `Q_CLASSINFO` definitions in the simulation item class declarations.

Once a simulation item hierarchy is in place, the same underlying data can be used to reverse the process and write down the configuration in a human-readable form. In Fig. 16 the *Creator* object is now replaced by a *Consumer* object that recursively visits the items in the hierarchy to produce the corre-

sponding output, using the information supplied by the *item discovery* module and the property handlers it spawns. Most importantly, SKIRT uses the `XmlHierarchyWriter` object to output a ski file (Fig. 6) after the user configured a simulation item hierarchy through a query and answer session (Fig. 5). A newly generated ski file is also stored with each set of simulation results, as a standard reference, explicitly listing the default values for properties that may have been omitted in the input ski file. Using the `LatexHierarchyWriter` object, SKIRT also writes a \LaTeX source file that documents the configuration in an even more user-friendly format (Fig. 7).

Because of this automation, adding a new SKIRT feature is extremely straightforward. For example, to add a new geometry, a developer would copy one of the existing geometry classes, rename the class, adjust the implementation and documentation of the member functions, adjust the `Q_CLASSINFO` definitions in the class declaration, and add a single line in the `RegisterSimulationItems` class to register the new geometry to the discovery system. Except for this trivial registration requirement, all information about the new geometry is in a single place, and the user interface will be automatically adjusted to incorporate it.

4. Conclusions

We described the major features of SKIRT, a state-of-the-art Monte Carlo dust radiation transfer simulation code used to study spiral galaxies, accretion disks and other astrophysical systems. In addition to its core capability of tracing the radiation through the dust, SKIRT offers a large number of built-in options for configuring all aspects of the simulation model, including spatial and spectral distributions, dust grain characterizations, simulated detection systems, and discretization.

Providing a proper user interface to support these complex configuration requirements is a nontrivial undertaking. Most SKIRT simulations run for hours or days, often on remote servers, so there is no need for a fancy graphical user interface. Still, the typical user is an expert astrophysicist who interacts with the SKIRT code rather occasionally, and thus benefits greatly from a low-barrier interface. SKIRT addresses this challenge through the combination of a *wizard*-like query and answer session to guide a first-time user through the configuration process, and self-documenting XML-based parameter files that can be easily updated in a text editor.

We further described the overall architecture of the code. Inspired by standard software design principles and patterns, the latest version of SKIRT has a modular implementation that can be easily maintained and expanded. Programming interfaces between components are well defined and narrow. The user interface is automatically constructed from data provided in the C++ class declarations, allowing a single point of definition, and placing the user interface information right next to the code implementing the corresponding feature.

All too often, scientific codes are written without much concern for user interface or for modular software design. This is very unfortunate. Scientists may not need a *graphical* user interface, but, just like every one else, they do benefit from an in-

teraction mechanism that hides the underlying complexity. As we have illustrated in this work, a well-designed non-graphical user interface may be a perfect fit, and can often be developed and maintained with limited resources. Similarly, adhering to proven software design principles pays off, even for small and mid-sized projects.

The SKIRT source code is publicly available, and it has already been applied to radiative transfer problems in various astrophysical domains. We welcome new applications, and we invite potential users and code contributors to join the SKIRT community.

Acknowledgements

This work fits in the CHARM framework (Contemporary physical challenges in Heliospheric and Astrophysical Models), a phase VII Interuniversity Attraction Pole (IAP) program organized by BELSPO, the BELgian federal Science Policy Office.

SKIRT and FitSKIRT are based on the Qt development framework and use the core Qt libraries for run-time class introspection, parallelization, string and container handling, and more. The code further incorporates the following third-party software libraries: CFITSIO developed by NASA's HEASARC to output FITS files; Voro++ described in Rycroft (2009) to help construct Voronoi dust grids; GALIB described in Wall (1996) to implement the search mechanism based on genetic algorithms.

The authors hereby thank all SKIRT users for their enthusiastic feedback, including many ideas for improvements and additions to the code.

References

References

- Baes, M., Davies, J.I., Dejonghe, H., Sabatini, S., Roberts, S., Evans, R., Linder, S.M., Smith, R.M., de Blok, W.J.G., 2003. Radiative transfer in disc galaxies - III. The observed kinematics of dusty disc galaxies. *MNRAS* 343, 1081–1094. doi:10.1046/j.1365-8711.2003.06770.x, arXiv:arXiv:astro-ph/0304501.
- Baes, M., Fritz, J., Gadotti, D.A., Smith, D.J.B., Dunne, L., da Cunha, E., Amblard, A., Auld, R., Bendo, G.J., Bonfield, D., Burgarella, D., Buttiglione, S., Cava, A., Clements, D., Cooray, A., Dariush, A., de Zotti, G., Dye, S., Eales, S., Frayer, D., Gonzalez-Nuevo, J., Herranz, D., Ibar, E., Ivison, R., Lagache, G., Leeuw, L., Lopez-Caniego, M., Jarvis, M., Maddox, S., Negrello, M., Michałowski, M., Pascale, E., Pohlen, M., Rigby, E., Rodighiero, G., Samui, S., Serjeant, S., Temi, P., Thompson, M., van der Werf, P., Verma, A., Vlahakis, C., 2010. Herschel-ATLAS: The dust energy balance in the edge-on spiral galaxy UGC 4754. *A&A* 518, L39. doi:10.1051/0004-6361/201014644, arXiv:1005.1773.
- Baes, M., Verstappen, J., De Looze, I., Fritz, J., Saftly, W., Vidal Pérez, E., Stalevski, M., Valcke, S., 2011. Efficient Three-dimensional NLTE Dust Radiative Transfer with SKIRT. *ApJS* 196, 22. doi:10.1088/0067-0049/196/2/22, arXiv:1108.5056.
- Bruzual, G., Charlot, S., 2003. Stellar population synthesis at the resolution of 2003. *MNRAS* 344, 1000–1028. doi:10.1046/j.1365-8711.2003.06897.x, arXiv:astro-ph/0309134.
- Camps, P., Baes, M., Saftly, W., 2013. Using 3D Voronoi grids in radiative transfer simulations. *A&A* 560, A35. doi:10.1051/0004-6361/201322281, arXiv:1310.1854.
- Cappellari, M., 2002. Efficient multi-Gaussian expansion of galaxies. *MNRAS* 333, 400–410. doi:10.1046/j.1365-8711.2002.05412.x, arXiv:astro-ph/0201430.

- Cashwell, E.D., Everett, C.J., 1959. A practical manual on the Monte Carlo method for random walk problems. International tracts in computer science and technology and their application, Oxford: Pergamon Press.
- Ciotti, L., Bertin, G., 1999. Analytical properties of the $R^{1/m}$ law. *A&A* 352, 447–451. [arXiv:astro-ph/9911078](#).
- Compiegne, M., Verstraete, L., Jones, A., Bernard, J.P., Boulanger, F., Flagey, N., Le Bourlot, J., Paradis, D., Ysard, N., 2011. The global dust SED: tracing the nature and evolution of dust with DustEM. *A&A* 525, A103. doi:10.1051/0004-6361/201015292, [arXiv:1010.2769](#).
- De Geyter, G., Baes, M., Camps, P., Fritz, J., De Looze, I., Hughes, T.M., Viaene, S., Gentile, G., 2014. The distribution of interstellar dust in CALIFA edge-on galaxies via oligochromatic radiative transfer fitting. *MNRAS* 441, 869–885. doi:10.1093/mnras/stu612, [arXiv:1403.7527](#).
- De Geyter, G., Baes, M., Fritz, J., Camps, P., 2013. FitSKIRT: genetic algorithms to automatically fit dusty galaxies with a Monte Carlo radiative transfer code. *A&A* 550, A74. doi:10.1051/0004-6361/201220126, [arXiv:1212.0538](#).
- De Looze, I., Baes, M., Bendo, G.J., Ciesla, L., Cortese, L., de Geyter, G., Groves, B., Boquien, M., Boselli, A., Brondeel, L., Cooray, A., Eales, S., Fritz, J., Galliano, F., Gentile, G., Gordon, K.D., Hony, S., Law, K.H., Madden, S.C., Sauvage, M., Smith, M.W.L., Spinoglio, L., Verstackpen, J., 2012a. The dust energy balance in the edge-on spiral galaxy NGC 4565. *MNRAS* 427, 2797–2811. doi:10.1111/j.1365-2966.2012.22045.x, [arXiv:1209.2636](#).
- De Looze, I., Baes, M., Fritz, J., Verstackpen, J., 2012b. Panchromatic radiative transfer modelling of stars and dust in the Sombrero galaxy. *MNRAS* 419, 895–903. doi:10.1111/j.1365-2966.2011.19759.x, [arXiv:1109.0212](#).
- Dehnen, W., 1993. A Family of Potential-Density Pairs for Spherical Galaxies and Bulges. *MNRAS* 265, 250.
- Dejonghe, H., 1987. A completely analytical family of anisotropic Plummer models. *MNRAS* 224, 13–39.
- Draine, B.T., Li, A., 2001. Infrared Emission from Interstellar Dust. I. Stochastic Heating of Small Grains. *ApJ* 551, 807–824. doi:10.1086/320227, [arXiv:astro-ph/0011318](#).
- Draine, B.T., Li, A., 2007. Infrared Emission from Interstellar Dust. IV. The Silicate-Graphite-PAH Model in the Post-Spitzer Era. *ApJ* 657, 810–837. doi:10.1086/511055, [arXiv:astro-ph/0608003](#).
- Einasto, J., 1965. On the Construction of a Composite Model for the Galaxy and on the Determination of the System of Galactic Parameters. *Trudy Astrofizicheskogo Instituta Alma-Ata* 5, 87–100.
- Emsellem, E., Monnet, G., Bacon, R., 1994. The multi-gaussian expansion method: a tool for building realistic photometric and kinematical models of stellar systems I. The formalism. *A&A* 285, 723–738.
- Fabian, D., Henning, T., Jäger, C., Mutschke, H., Dorschner, J., Wehrhan, O., 2001. Steps toward interstellar silicate mineralogy. VI. Dependence of crystalline olivine IR spectra on iron content and particle shape. *A&A* 378, 228–238. doi:10.1051/0004-6361:20011196.
- Fioc, M., Rocca-Volmerange, B., 1997. PEGASE: a UV to NIR spectral evolution model of galaxies. Application to the calibration of bright galaxy counts. *A&A* 326, 950–962. [arXiv:astro-ph/9707017](#).
- Gamma, E., Helm, R., Johnson, R., Vliissides, J., 1994. Design Patterns: Elements of Reusable Object-Oriented Software. 1st ed., Addison-Wesley Professional.
- Granato, G.L., Danese, L., 1994. Thick Tori around Active Galactic Nuclei - a Comparison of Model Predictions with Observations of the Infrared Continuum and Silicate Features. *MNRAS* 268, 235.
- Groves, B., Dopita, M.A., Sutherland, R.S., Kewley, L.J., Fischera, J., Leitherer, C., Brandl, B., van Breugel, W., 2008. Modeling the Pan-Spectral Energy Distribution of Starburst Galaxies. IV. The Controlling Parameters of the Starburst SED. *ApJS* 176, 438–456. doi:10.1086/528711, [arXiv:0712.1824](#).
- Jaeger, C., Molster, F.J., Dorschner, J., Henning, T., Mutschke, H., Waters, L.B.F.M., 1998. Steps toward interstellar silicate mineralogy. IV. The crystalline revolution. *A&A* 339, 904–916.
- Kurucz, R.L., 1993. Model Atmospheres (Kurucz, 1979). *VizieR Online Data Catalog* 6039, 0.
- Leitherer, C., Schaerer, D., Goldader, J.D., Delgado, R.M.G., Robert, C., Kune, D.F., de Mello, D.F., Devost, D., Heckman, T.M., 1999. Starburst99: Synthesis Models for Galaxies with Active Star Formation. *ApJS* 123, 3–40. doi:10.1086/313233, [arXiv:astro-ph/9902334](#).
- Lucy, L.B., 1999. Computing radiative equilibria with Monte Carlo techniques. *A&A* 344, 282–288.
- Maraston, C., 1998. Evolutionary synthesis of stellar populations: a modular tool. *MNRAS* 300, 872–892. doi:10.1046/j.1365-8711.1998.01947.x, [arXiv:astro-ph/9807338](#).
- Mathis, J.S., Rumpl, W., Nordsieck, K.H., 1977. The size distribution of interstellar grains. *ApJ* 217, 425–433. doi:10.1086/155591.
- Min, M., Hovenier, J.W., de Koter, A., 2005. Modeling optical properties of cosmic dust grains using a distribution of hollow spheres. *A&A* 432, 909–920. doi:10.1051/0004-6361:20041920, [arXiv:astro-ph/0503068](#).
- Misiriotis, A., Kylafis, N.D., Papamastorakis, J., Xilouris, E.M., 2000. Is the exponential distribution a good approximation of dusty galactic disks? *A&A* 353, 117–123. [arXiv:astro-ph/9912018](#).
- Netzer, H., 1987. Quasar discs. II - A composite model for the broad-line region. *MNRAS* 225, 55–72.
- Niccolini, G., Woitke, P., Lopez, B., 2003. High precision Monte Carlo radiative transfer in dusty media. *A&A* 399, 703–716. doi:10.1051/0004-6361:20021761.
- Plummer, H.C., 1911. On the problem of distribution in globular star clusters. *MNRAS* 71, 460–470.
- Retana-Montenegro, E., van Hese, E., Gentile, G., Baes, M., Frutos-Alfaro, F., 2012. Analytical properties of Einasto dark matter haloes. *A&A* 540, A70. doi:10.1051/0004-6361/201118543, [arXiv:1202.5242](#).
- Rycroft, C.H., 2009. Voro++: A three-dimensional voronoi cell library in c++. *Chaos* 19, 041111; doi: 10.1063/1.3215722.
- Saftly, W., Baes, M., Camps, P., 2014. Hierarchical octree and k-d tree grids for 3D radiative transfer simulations. *A&A* 561, A77. doi:10.1051/0004-6361/201322593, [arXiv:1311.0705](#).
- Saftly, W., Camps, P., Baes, M., Gordon, K.D., Vandewoude, S., Rahimi, A., Stalevski, M., 2013. Using hierarchical octrees in Monte Carlo radiative transfer simulations. *A&A* 554, A10. doi:10.1051/0004-6361/201220854, [arXiv:1304.2896](#).
- Schartmann, M., Meisenheimer, K., Camenzind, M., Wolf, S., Henning, T., 2005. Towards a physical model of dust tori in Active Galactic Nuclei. Radiative transfer calculations for a hydrostatic torus model. *A&A* 437, 861–881. doi:10.1051/0004-6361:20042363, [arXiv:astro-ph/0504105](#).
- Sérsic, J.L., 1963. Influence of the atmospheric and instrumental dispersion on the brightness distribution in a galaxy. *Boletín de la Asociación Argentina de Astronomía La Plata* Argentina 6, 41.
- Stalevski, M., 2012. SKIRTOR - database of modelled AGN dusty torus SEDs. *Bulgarian Astronomical Journal* 18, 030000.
- Stalevski, M., Fritz, J., Baes, M., Nakos, T., Popović, L.Č., 2012. 3D radiative transfer modelling of the dusty tori around active galactic nuclei as a clumpy two-phase medium. *MNRAS* 420, 2756–2772. doi:10.1111/j.1365-2966.2011.19775.x, [arXiv:1109.1286](#).
- Steinacker, J., Baes, M., Gordon, K.D., 2013. Three-Dimensional Dust Radiative Transfer*. *ARA&A* 51, 63–104. doi:10.1146/annurev-astro-082812-141042, [arXiv:1303.4998](#).
- Suto, H., Sogawa, H., Tachibana, S., Koike, C., Karoji, H., Tsuchiyama, A., Chihara, H., Mizutani, K., Akedo, J., Ogiso, K., Fukui, T., Ohara, S., 2006. Low-temperature single crystal reflection spectra of forsterite. *MNRAS* 370, 1599–1606. doi:10.1111/j.1365-2966.2006.10594.x.
- Tremaine, S., Richstone, D.O., Byun, Y.I., Dressler, A., Faber, S.M., Grillmair, C., Rikhardy, J., Lauer, T.R., 1994. A family of models for spherical stellar systems. *AJ* 107, 634–644. doi:10.1086/116883, [arXiv:astro-ph/9309044](#).
- van der Kruit, P.C., 1986. Surface photometry of edge-on spiral galaxies. V - The distribution of luminosity in the disk of the galaxy derived from the Pioneer 10 background experiment. *A&A* 157, 230–244.
- Verstackpen, J., Fritz, J., Baes, M., Smith, M.W.L., Allaert, F., Bianchi, S., Blommaert, J.A.D.L., De Geyter, G., De Looze, I., Gentile, G., Gordon, K.D., Holwerda, B.W., Viaene, S., Xilouris, E.M., 2013. HERschel Observations of Edge-on Spirals (HEROES). I. Far-infrared morphology and dust mass determination. *A&A* 556, A54. doi:10.1051/0004-6361/201220733, [arXiv:1305.3130](#).
- Voronoi, G., 1908. Nouvelles applications des paramètres continus à la théorie de formes quadratiques. deuxième mémoire. recherches sur les parallélogrammes primitifs. *Journal für die reine und angewandte Mathematik* 134, 198–287.
- Wall, M., 1996. GALib: A C++ library of genetic algorithm components.

- PhD thesis, Mechanical Engineering Department, Massachusetts Institute of Technology .
- Weingartner, J.C., Draine, B.T., 2001. Dust Grain-Size Distributions and Extinction in the Milky Way, Large Magellanic Cloud, and Small Magellanic Cloud. *ApJ* 548, 296–309. doi:10.1086/318651, arXiv:astro-ph/0008146.
- Whitney, B.A., 2011. Monte Carlo radiative transfer. *Bulletin of the Astronomical Society of India* 39, 101–127. arXiv:1104.4990.
- Yusef-Zadeh, F., Morris, M., White, R.L., 1984. Bipolar reflection nebulae - Monte Carlo simulations. *ApJ* 278, 186–194. doi:10.1086/161780.
- Zubko, V., Dwek, E., Arendt, R.G., 2004. Interstellar Dust Models Consistent with Extinction, Emission, and Abundance Constraints. *ApJS* 152, 211–249. doi:10.1086/382351, arXiv:astro-ph/0312641.